# Extending Online Travel Agency with Adaptive Reservations

# Extending Online Travel Agency with Adaptive Reservations

Yu Zhang[1], Wenfei Fan[2], Huajun Chen[1], Hao Sheng[1], and Zhaohui Wu[1]

[1] College of Computer Science,
Zhejiang University, Hangzhou 310027, Zhejiang, China
{yzh,huajunsir,wzh}@zju.edu.cn
[2] University of Edinburgh & Bell Laboratories
wenfei@inf.ed.ac.uk

**Abstract.** Current online ticket booking systems either do not allow customers to reserve a ticket with a locked price, or grant a fixed reservation timespan, typically 24 hours. The former often leads to *false availability*: when a customer decides to purchase a ticket after a few queries, she finds that either the ticket is no longer available or the price has hiked up. The latter, on the other hand, may result in unnecessary *holdback*: a customer cannot purchase a ticket because someone else is holding it, who then cancels the reservation after an excessively long period of time. False availability and holdback routinely lead to loss of revenues, credibility and above all, customers. To rectify these problems, this paper introduces a transaction model for e-ticket systems to support a reservation functionality: customers can reserve tickets with a locked price, for a timespan that is determined by the demands on the tickets, rather than being fixed for all kinds of the tickets. We propose a method for implementing the model, based on hypothetical queries and triggers. We also show how to adjust the reservation timespan *w.r.t.* demands. We experimentally verify that our model and methods effectively reduce both false availability and holdback rates. These yield a practical approach to improving not only e-ticket systems but also other e-commerce systems.

## 1 Introduction

It is increasingly common for people to book travel packages online. A number of online ticket booking systems (*a.k.a.* e-ticket systems or virtual travel agents) have been launched by airlines [1] or Web service providers (*e.g.,* Expedia [2], Orbitz [3] and Priceline [4]). While these services allow customers to query airfare and purchase tickets online, they provide very limited support for one to *reserve* a ticket with a locked price.

It is evident that customers want to *reserve* a ticket before they are ready to purchase it. For instance, Alice wants to book a ticket via an online service. After finding a ticket with a reasonable price, she proceeds to build up the rest of her travel plan by, *e.g.,* issuing queries about hotels and car rental. After 5 minutes, she is happy with the package she found and decides to purchase the

ticket. But she is frustrated to find that either the ticket she liked has already been sold out, or the price has gone up. She has to start the process again from scratch, and may get her travel package in place only after repeated failures. She would certainly like it if the system had allowed her to reserve the ticket with the price locked *when* she found it.

Existing e-ticket systems support reservations based on one of the following approaches.

(a) On one end of the spectrum, reservations are treated as "purchases": as soon as tickets are reserved, they are blocked from other customers *until* they are finally purchased or the reservation is canceled. The other customers cannot view or query these tickets as if they were already sold. However, typically certain percentage of the reservations will be canceled in the end. As a result, this *conservative* approach inevitably incurs excessive and unnecessary rollbacks and transaction management cost. Worse still, with this comes *holdback* of tickets: other customers are denied the chance to purchase those tickets that are reserved but finally *not* purchased. This often leads to loss of revenue, among other things.

(b) On the other end of the spectrum, reservations are treated as "queries": no reservation action is taken. This *aggressive* approach leads to *false availability*: a customer finds tickets with a reasonable price; however, when she is ready to commit to purchase, she is told that the tickets with the price are actually not available. While this approach does not suffer from the holdback problem, it leads to loss of credibility and eventually loss of customers.

(c) Compromising these two extreme approaches, some services allow one to reserve a ticket for a *fixed* period of time, typically 24 hours. That is, a reservation will expire after 24 hours if no explicit purchase or cancellation is conducted. This reduces holdback and false availability rates, but only to an extent: in a travel season when certain tickets are highly demanded, a fixed 24-hour reservation timespan is often excessively long and may incur the same holdback problem as approach (a) above. In practice the reservation timespan should be *adaptive*, *i.e.,* it should *vary* in accordance to the demands on the tickets. Furthermore, it is common that customers decide to commit to purchase in a single session of querying and purchasing; thus a fixed 24-hour timespan is often an overkill.

We have investigated 30 popular e-ticket systems, and found that all except seven adopt approach (b), *i.e.,* no reservation function is supported at all. The seven services that support reservation all follow approach (c) (American Airline, Continental, Southwest, United, and US Airways [1], MakemyTrip.com and Cfares.com): customers are allowed to hold the tickets for 24 hours or until the midnight of the next day. In addition to the holdback problem with approach (c), these services do *not* allow the price of a reserved ticket to be locked; thus although a customer can reserve a ticket she is not guaranteed to get the ticket with the price she found when making the reservation. In short, the reservation functionality supported by existing online ticket booking systems neither rectifies the holdback problem nor reduces false availability.

**Contributions.** To this end we propose a transaction model for e-ticket systems to support the reservation functionality while reducing both the holdback rate and the false availability rate. In this model the customer is allowed to reserve a ticket, with a *locked* price, for a timespan adjusted in accordance to demands on the tickets. We also provide an efficient technique to support the model such that the addition of the reservation function does not imply any drastic degradation in performance for existing e-ticket systems.

The main contributions of the paper include the following:

1. A transaction model that supports queries, purchases and in addition, *reservations*. It reduces the holdback and false availability rates by (a) adapting reservation timespan based on demands, and (b) making a percentage of reserved tickets available to other customers, determined by an estimate of the reservations that eventually turn into purchases.
2. A combination of techniques, including hypothetical queries (see, *e.g.,* [5,6]) and triggers (see, *e.g.,* [7,8]), to efficiently support the transaction model.
3. A method for computing the reservation timespan based on demands on different tickets.
4. A preliminary experimental study that verifies the effectiveness and efficiency of our techniques.

These will provide online travel systems with a practical method to support the reservation functionality. We should remark that although we focus on ticket booking systems to simplify the discussion, the techniques proposed in this work are generic enough to be applicable to other e-commerce systems such as finance services.

**Organization.** We introduce our transaction model and implementation technique, as well as a method for computing reservation timespan in Section 2. Our experimental results are presented in Section 3, followed by related work and future work in Section 4.

## 2 A Transaction Model

In this section we first present the transaction model. We then present techniques for efficiently implementing the model. Finally, we outline the architecture of an e-ticket system based on this model.

### 2.1 Supporting Reservations

We propose an e-ticket system that allows customers to reserve tickets, in addition to querying and purchasing. A customer may reserve tickets with a locked price, for a timespan $s$ computed by the system based on demands on the tickets. From the time when the reservation is granted until the end of the timespan $s$, the customer may either (a) commit to purchase the tickets or (b) cancel the reservation. A short while before the end of $s$, the system sends a reminder to

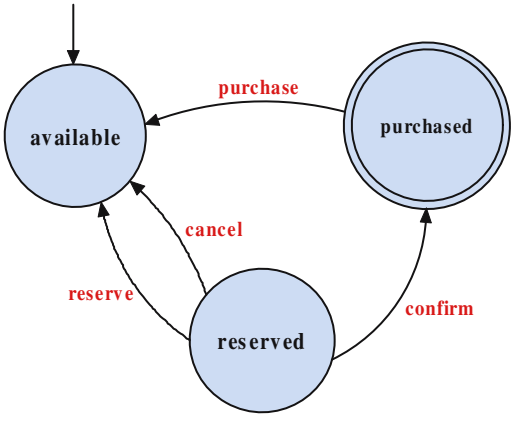


**Fig. 1.** FSM representation of transactions of e-Ticket System

the customer about the reservation; if no action is taken by the customer, the system cancels the reservation at the end of $s$, *i.e.,* when the reservation *expires*.

In the transaction model for the e-ticket system, each ticket is associated with one of following *states*: available, reserved or purchased. The state of a ticket changes from available to reserved when the ticket is reserved by a customer. It in turn changes from reserved to purchased if the customer commits to purchase, and to available if the reservation is canceled either by the customer or by the system when the reservation expires. While transition may take place from available to purchased, it cannot change from purchased to available or reserved. More formally, this can be characterized in terms of a (deterministic) finite state machine (FSM), which is commonly used in modeling Web services (see, *e.g.,* [9,10]).

FSM=($\Sigma$,$S$,$s_0$,$\Delta$,$F$), where:

- $\Sigma = \{reserve, cancel, confirm, purchase\}$
- $S = \{$available, reserved, purchased$\}$
- $s_0 = \{$available$\}$
- $\Delta = S \times \Sigma \rightarrow S$
- $F = \{$purchased$\}$

Figure 1 shows an abstract representation as FSM of the reservation process. We refer to the set $\tau$ of tickets of each flight (train, car, etc) as tickets of *type* $\tau$.

- We say that tickets of type $\tau$ are *held-back* if when a customer wants to purchase or reserve a ticket of type $\tau$, all tickets of $\tau$ are either purchased or reserved at the moment, and moreover, some of those reserved tickets change to available later on.
- We say that tickets of $\tau$ are *falsely available* if a customer is allowed to reserve a ticket of $\tau$ but later on cannot purchase it (*i.e.,* change its state from reserved to purchased).

Let $N$ denote the number of successful reservations made on tickets of $\tau$ that eventually change from reserved to purchased, $n_h$ the number of failed reserva-

tions when tickets of $\tau$ are held-back, and $n_f$ the number of reservations on tickets that are falsely available. We define the false availability rate and holdback rate to be $n_f/(N+n_f)$ and $n_h/(N+n_h)$, denoted by $\delta$ and $\gamma$, respectively.

We aim to reduce both $\delta$ and $\gamma$. Since it is not always possible to minimize both the false availability rate and the holdback rate, we give higher priority to reducing the false availability rate since it typically inflicts more severe damages when credibility and customers are concerned. Below we propose two methods to reduce false availability rate $\delta$ while keeping the the holdback rate $\gamma$ low.

First, we introduce a parameter $\alpha$ in the range $[0, 1]$, referred to as the *purchase rate*, for the set of tickets of each type $\tau$. The purchase rate indicates (an estimate of) the percentage of the tickets of which the states change from reserved to purchased. Intuitively, we make $(1 - \alpha)$ percent of reserved tickets available to customers, so that not all "hot" tickets would be held back. Observe that the smaller $\alpha$ is, the less the holdback rate is, but on the other hand, the higher the false availability rate is.

Second, we assign different reservation timespan $s$ to different types of tickets. In Section 2.2, we shall present a method to compute $s$ based on demands on tickets of type $\tau$. Intuitively, the smaller $s$ is, the less the holdback rate is. When it comes to the false availability rate $\delta$, the story is a bit more complicated. Making $s$ larger may on one hand hold the ticket longer so that when the customer decides to commit to the purchase, the ticket will still be available; but on the other hand, if the purchase rate $\alpha$ is small, the chances are that the reserved tickets become unavailable and thus it may leads to higher $\delta$. As will be seen in Section 3, making $s$ larger may reduce $\delta$ only if $\alpha$ is sufficiently large.

For the reservation approaches adopted by existing e-ticket systems surveyed in Section 1, approach (b) adopts $\alpha = 0$ and $s$ does not exist, while approaches (c) uses $\alpha = 1$ and fixes $s$ to be 24 hours. In our model, we set $\alpha$ to a value between 0 and 1, determined by statistical analysis and estimate, and will be seen in Section 2.2, we compute $s$ based on demands on tickets of type $\tau$ rather than giving a fixed timespan for all types of tickets. As will be seen in Section 3, we keep $\alpha$ and $s$ sufficiently large so that the false availability rate remains low while the holdback rate is reduced.

### 2.2 Implementation Techniques

A naive approach to implementing reservations is to create and maintain relations for storing information about available, reserved and purchased tickets, and, whenever the state of a ticket changes from reserved to purchased (resp. available), or the other way around, we modify both the reserved and the purchased (resp. available) tables. This, however, incurs heavy transaction cost. In light of this, we propose a technique based on hypothetical queries to reduce the overhead. We also present a method to compute the reservation timespan based on demands on different tickets.

**Relations for Reserved, Purchased and Available Tickets.** Along the same lines as existing e-ticket systems, we store information about tickets of various states in *fact* tables:

- Ticket table $T(\mathsf{id}, \sharp, \mathsf{price})$ keeps track of the number $\sharp$ of available tickets of a specific flight $\mathsf{id}$.
- Reservation table $R$ $(\mathsf{id}, \sharp, \mathsf{t_s}, \mathsf{price}, \mathsf{info})$ stores reservations made so far. A customer reserves $\sharp$ tickets on flight $\mathsf{id}$ with a locked $\mathsf{price}$, where $t_s$ is a timestamp specifying when the reservation expires, and $\mathsf{info}$ denotes some basic information about the customer such as name, nationality, number of tickets, etc.
- Purchase table $P(\mathsf{id}, \sharp, \mathsf{price}, \mathsf{info})$ stores the *real* purchases.
- Other tables store other information about the flight such as departure time, arrival time and destinations, etc.

**Hypothetical Queries.** A *hypothetical query* is of the form $Q$ when $\{\{U\}\}$, where $U$ is an update (see [6]). It to find the value that query $Q$ would return on a database $DB$ that would be obtained by executing update $U$ on the original $DB$, *without* actually updating $DB$.

We regard a reservation as a *hypothetical purchase* while final commitment to purchase as a *real purchase*. Capitalizing on hypothetical queries, when a reservation is made, we *only* need to modify the reservation table $R$, *without* changing either $T$ or $P$. We modify $T$, $R$ and $P$ only when a reservation is converted into a real purchase. This reduces the overhead of unnecessary transactions and rollbacks.

To carry this out, for each query $Q$ on $T$, we automatically rewrite it into a hypothetical query $Q_T = Q$ when $\{\{U\}\}$ on both $T$ and $R$, such that $\alpha$ percent of the reservations in $R$ are "taken out" from $T$ during the process, where $\alpha$ is the purchase rate given earlier. More specifically, the update $U$ is of the form:

$$U ::= del(T, \alpha R) \quad \text{delete } \alpha \cdot \sharp \text{ tickets of } R \text{ from } T$$

In a nutshell, for each occurrence of $T$ in $Q$, we replace it with $del(T, \alpha R)$ by taking out certain tickets already reserved. Thus

$$Q_T = Q \text{ when } \{\{del(T, \alpha R)\}\} \tag{1}$$

As remarked earlier, we make $(1 - \alpha)$ tickets in $R$ available to customers to reduce the holdback rate, since typically only $\alpha$ percent of reservations will lead to real purchases in the end.

A number of techniques have been developed for efficiently evaluating hypothetical queries. Here we adopt the lazy approach of [6]. More specifically, we first rewrite $Q_T$ into an equivalent, non-hypothetical query $Q_T^{'}$ by transforming each $U$ into an "explicit substitution", and then applying the substitution and obtaining a pure SQL query. We illustrate this by using a hypothetical query of form (1), where $Q$ is an SQL query for finding the number of tickets in stock for flight $k$ with price $p$, *i.e.,*

$$Q = \pi_{\sharp}(\sigma_{\mathsf{id}=k \wedge \mathsf{price}=p}\, T). \tag{2}$$

We replace the update with explicit substitution:

$$\pi_{\sharp}(\sigma_{\mathsf{id}=k\wedge\mathsf{price}=p}\, T) \text{ when } \{\{(T - \alpha R)/T\}\}$$

Note that $R$ is a bag of records since for each flight there may be several reservations for it by different customers. Thus to deduct the total number of reserved tickets in $R$, we need to use aggregate function $\mathsf{sum}$. Now we apply the substitution to the query $Q_T$ and get

$$Q_T^{'} \equiv \pi_{\sharp}(\sigma_{\mathsf{id}=k\wedge\mathsf{price}=p}\, T) - \alpha\, \mathsf{sum}(\pi_{\sharp}(\sigma_{\mathsf{id}=k\wedge\mathsf{price}=p} R))$$

This query is equivalent to $Q_T$. Similarly we can automatically rewrite other queries, *e.g.,* queries for finding airfare.

The use of hypothetical queries and automated query rewriting allows us not to update $T$ and $P$ when making or canceling a reservation, and thus reduce the overhead of transactions.

**Making, Canceling and Committing Reservations.** As remarked earlier, there is no need to update the ticket and purchase tables $T$ and $P$ when a reservation is made or canaled. Indeed, only the reservation table $R$ needs to be changed in response to these updates. In contrast, all three tables $T, R$ and $P$ are updated when a user commits to purchase a reserved ticket.

**Making a reservation.** Upon receiving a customer request for reserving $n$ tickets for flight $k$ with a locked price $p$, the system does the following, in *one transaction.* (*i*) It generates a query $Q$ of form (2) given above, which is to find the number of available tickets for flight $k$ with price $p$. The query $Q$ is rewritten into a *hypothetical query* of form (1) above, and is evaluated on the ticket table $T$ and the reservation table $R$ using the evaluation techniques described above. If the result of the query $Q$ is negative, then the customer request is *denied.* Otherwise the following steps are taken. (*ii*) It computes the reservation timespan $s$ and based on $s$, timestamp $t_s = t + s$, where $t$ is the current time. We will show how $s$ is computed shortly. (*iii*) It inserts a tuple $(\mathsf{id} = k, \sharp = n, \mathsf{price} = p, \mathsf{info} = i)$ into the reservation table $R$, where $i$ is the related customer information. Note that neither table $T$ nor table $P$ is updated.

**Canceling a reservation.** When a customer requests to cancel a reservation of $n$ tickets for flight $k$, the systems finds the corresponding tuple from the reservation table $R$, based on both the flight and customer information. It then removes the tuple from $R$. No other tables are updated.

**Committing to purchase a reserved ticket.** When a customer commits to purchase reserved tickets, the system does the following, in *one* transaction. (*i*) It first identifies the corresponding tuple $r = (\mathsf{id} = k, \sharp = n, \mathsf{price} = p, \mathsf{info} = i)$ from the reservation table $R$, and removes $r$ from $R$. (*ii*) It then forms and evaluates query $Q$ as in step (*i*) for making reservations. If the result of $Q$ is negative, then the tickets are *falsely available* and the transaction aborts. Otherwise the system proceeds to do the following. (*iii*) It inserts $r$ into purchase table $P$.

(*iv*) It also updates ticket table $T$ by removing $n$ tickets of flight $k$ from $T$. Upon the completion of the transaction, the hypothetical purchase of this reservation becomes a *real* purchase. Note that only at this stage all three tables $T, R$ and $P$ need to be modified.

A subtle issue arises when the price of tickets for flight $k$ in the ticket table $T$ is updated. To keep the price of the reserved tickets for flight $k$ unchanged, the system does the following. (*i*) It first finds the total number $n$ of reserved tickets for flight $k$ in the reservation table. (*ii*) In the ticket table $T$, it keeps the price of $\alpha \cdot n$ tickets for flight $k$ unchanged, while updating the price of the remaining tickets for flight $k$, where $\alpha$ is the purchase rate.

**Adapting Reservation Timespan.** As remarked earlier, we determine the reservation timespan for different tickets based on demands on the tickets. More specifically, for tickets of each type $\tau$, we characterize the demands on the tickets using the following parameters.

- *weight* $w$ in the range $[0, 1]$, indicating the "popularity" of the flight; this is determined by statistical analysis of the historical data of the flight, and may vary in different travel seasons; indeed, the demand for flights to Orlando is typically higher before Christmas than that during school terms;
- *advance parameter* $d$, which is the number $d$ of days prior to the departure of the flight when the reservation is made;
- the *maximum timespan* $s_{\mathsf{max}}$; here one may use 24 hours as the default value, following the practice of most airlines.

We use the following simple formula to compute the reservation timespan $s$ based on these parameters:

$$s = (1 - w) \cdot f(d) \cdot s_{\mathsf{max}}, \tag{3}$$

where $f(d)$ is a function in which $c$ is a constant in $[0, 1]$:

$$f(d) = \begin{cases} c \cdot d/14 \text{ if } d \leq 14 \\ c \qquad\quad \text{otherwise} \end{cases}$$

Intuitively, the more popular the flight is, the less reservation timespan $s$ is; furthermore, the less days in advance the reservation is made, the less $s$ is. In function $f(d)$ we choose constant 14 in accordance to the common practice of most airlines: "penalty" is incurred if the reservation is made within two weeks prior to the scheduled departure of the flight.

**Triggers for Expiring Reservations.** Shortly before a reservation expires, the system sends the customer a reminder. Recall that when a customer reserves a ticket, a reservation timespan $s$ and a timestamp $t_s$ are computed and stored in the corresponding tuple in the reservation table. That is, the reservation remains valid for a period $s$ of time until time $t_s$. Meanwhile, a *trigger* is set up such that shortly before the reservation expires, say 30 minutes before $t_s$, action will be triggered to generate the reminder and notify the customer. If the customer takes no action before the reservation expires, the system performs the cancellation operation at time $t_s$, as described above.

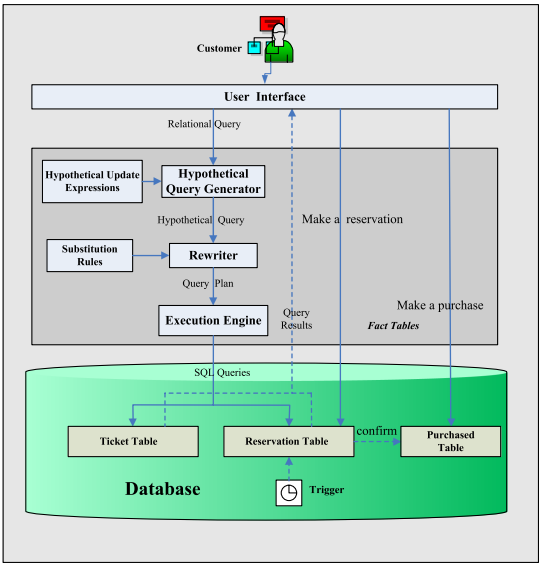


**Fig. 2.** The Architecture of an e-Ticket System

### 2.3 The Architecture of a Ticket Booking System

Putting these together, we propose to develop a 3-tier ticket booking system based on our transaction model, as depicted in Figure 2. The top layer is the user interface for customers to book tickets, the bottom layer is the underlying database storing *fact* tables, and the middle-tier processes customer queries, reservation requests and purchase orders. More specifically, upon receiving a customer query, the middle-tier converts it into an equivalent hypothetical query and evaluates it following the strategy given in Section 2.2. Upon receiving a request for making a reservation, the middle-tier starts a transaction to process it as described in Section 2.2. In particular, it computes the reservation timespan using the formula given in Section 2.2. The customer may succeed in making the reservation if there are enough tickets available. Subsequently the customer may cancel the reservation or commit to purchase the reserved tickets, which are again processed using the strategies given in Section 2.2. Furthermore, when a reservation is made, a trigger is set up, such that the customer will be notified shortly before the reservation expires, as described in Section 2.2.

## 3 Experimental Study

Our experimental evaluation focuses on the effectiveness of our reservation model in reducing the holdback and false availability rates $\delta$ and $\gamma$. We compare our approach against the approaches adopted by existing e-ticket systems, namely, approaches (b) and (c) described in Section 1, referred to as the *no-hold* and

*fully-hold* approaches, respectively. We also investigate the impact of the purchase parameter $\alpha$ and reservation timespan $s$ on $\delta$ and $\gamma$.

### 3.1 Experimental Setting

We considered a flight with up to 350 seats initially available. We randomly generated a set of reservations and subsequent confirmations or cancellations. Each reservation requested $n$ tickets, where $n$ is a number randomly chosen from [1,5]. We assumed that the flight was popular: there were sufficiently many customers to query about and book tickets until all the tickets would be sold out. In a duration of 30 days, we assumed that the arrival of customers followed a Poisson process (see, *e.g.,* [11]) and customer arrival rate is set to $\lambda = 1$ unless other specified. The intervals between successive customer arrivals were treated as independent random variables. The latency between a reservation and its subsequent confirmation or cancellation was also generated randomly within time slot [0, $s$]; that is, we assumed that most of customers will inspect their reservations within the timespan $s$ offered by the e-ticket system. For each reservation, we generated another random value to determine whether or not the reservation is confirmed or canceled. Unless specified otherwise, we fixed the confirmation rate to 70% and customer arrival rate to $\lambda = 1$.

The experiments were run on a machine with a 2.40GHZ Pentium IV processor and 512MB of RAM. Each experiment was run 200 times and the average is reported here.

### 3.2 Experimental Results

we study the impact of the purchase rate $\alpha$ and reservation timespan $s$ on reducing the false availability and holdback rates $\delta$ and $\gamma$. More specifically, with $s$ (resp. $\alpha$) fixed to certain values we investigate the effect of varying $\alpha$ (resp. $s$). The goal is twofold: first, we want to verify that the use of $\alpha$ and $s$ indeed reduces $\delta$ and $\gamma$; second, we want to study the behavior of different $s$ and $\alpha$, and find out what values we should choose for them. For the reservation timespan $s$ we either use fixed values or vary its values from 0 to 24 hours, rather than using formula (3) given in Section 2.2.

Recall notations $N, n_f$ and $n_h$ from Section 2.1. In each run, $N$ records the total number of reservations that lead to purchases, $n_h$ the total number of held-back reservations and $n_f$ the total number of reservations on falsely available tickets (*i.e.,* the reservations that will not turn into real purchases when customers come back and inspect their reserved tickets, before or after the reservations expire), in the *entire* life-cycle of ticket selling of the flight. The average of 200 runs is reported.

**The effect of varying the purchase rate $\alpha$.** Fixing reservation timespan $s$ to be 6, 12 and 24 hours, we vary the purchase rate $\alpha$ and study its effect on the false availability and holdback rates $\delta$ and $\gamma$. Observe that when $\alpha = 0$, it characterizes the *no-hold* approach, and on the other hand, when $\alpha = 1$, it corresponds to the *fully-hold* approach. As Figures 3 and 4 show, as expected,

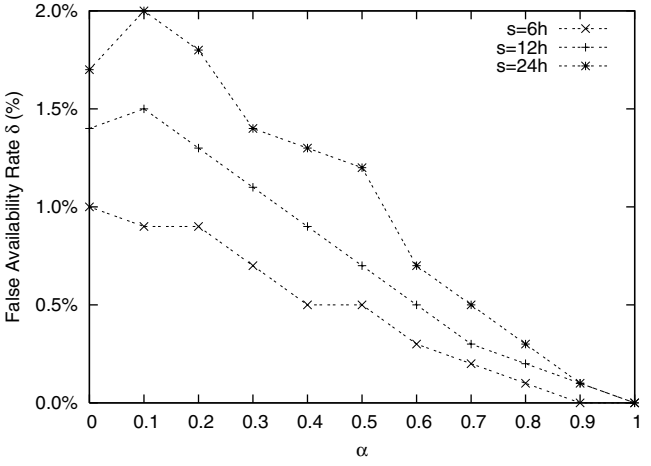


**Fig. 3.** Effect of varying $\alpha$ on the false availability rate($s = 5, 12, 24$ hours)

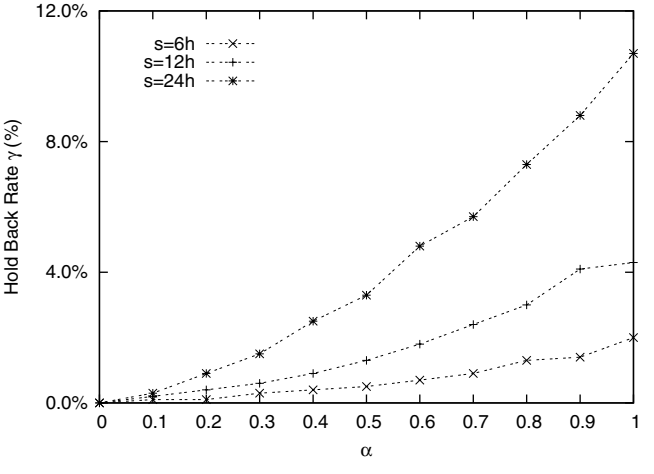


**Fig. 4.** Effect of varying $\alpha$ on the holdback rate ($s = 5, 12, 24$ hours)

when $\alpha$ increases the holdback rate increases while the false availability rate decreases. It also tells us that while the *no-hold* approach does not have the holdback problem, it incurs a rather high false availability rate. On the other hand, while the *fully-hold* approach does not lead to false available reservations, its holdback rate is rather high. In contrast, if we choose $\alpha \geq 0.8$, the false availability rates reduce to a neglectable value when either $s = 6$, $s = 12$ or $s = 24$, while their holdback rates are lower than the *fully-hold* counterparts.

**The effect of varying the reservation timespan $s$.** Fixing $\alpha$ to be $0, 0.5, 0.8$ and 1, we vary $s$ from 0 to 24 hours and measure the false availability and holdback rates $\delta$ and $\gamma$ for each $s$. As shown in Figures 5 and 6, when $\alpha$ increases,

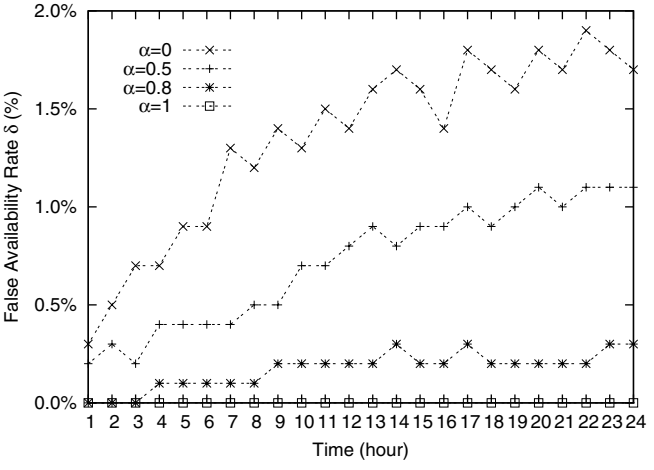


**Fig. 5.** Effect of varying $s$ on the false availability rate ($\alpha = 0, 0.5, 0.8, 1$)

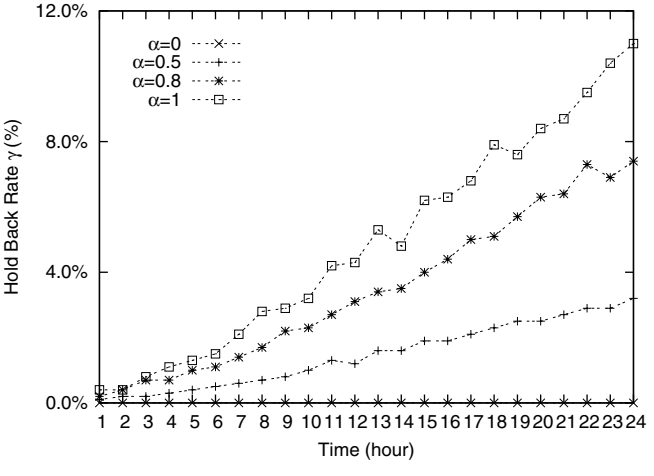


**Fig. 6.** Effect of varying $s$ on the holdback rate ($\alpha = 0, 0.5, 0.8, 1$)

the holdback rate $\gamma$ increases while the false availability rate $\delta$ decreases, as expected. One might be tempted to reduce timespan $s$ in order to minimize both false availability rate and holdback rate. However, this is not a practical solution. Reducing timespan means that the e-ticket system enforces an inadequate period of time for customers to reserve tickets, who may then find that the reservations expire after a short period of time, and cannot be very happy about it. Therefore, any practical e-ticket system should not use a very small timespan $s$. On the other hand, if the timespan is too large (which means customers are allowed to hold tickets for a long time), the holdback rate goes up; it is more likely that more customers are unable to purchase the tickets which are in fact available. This highlights the need for making $s$ adaptive to the demands on the tickets.

For the *no-hold* approach, *i.e.,* when $\alpha = 0$, $\gamma$ becomes 0 but $\delta$ is high. For the *fully-hold* approach, *i.e.,* when $\alpha = 1$, $\delta$ becomes 0 while $\gamma$ gets rather high. When $\alpha \geq 0.8$, the false availability rate $\delta$ is much lower than the *no-hold* approach while the holdback rate is also greatly reduced compared to the *fully-hold* counterpart.

**Discussion.** We have presented several results from our experimental study of our reservation model. First, the results verify that our approach clearly outperforms the *no-hold* and *fully-hold* approaches adopted by existing e-ticket systems. With a lower holdback rate compared with *fully-hold* model, it has neglectable false availability rate in contrast to the *no-hold* approach. Second, we find that when $\alpha \geq 0.8$, the false availability rate is almost 0 in all cases. These suggest how e-ticket systems may choose purchase rate $\alpha$ and adjust reservation timespan $s$.

## 4 Concluding Remarks

We have investigated the false availability and holdback problems in connection with existing online ticket booking systems. To rectify these we have proposed a transaction model that supports adaptive customer reservations with neglectable false availability and lower holdback rates. To efficiently implement the model we have developed a combination of techniques such as the analysis of purchase rate, hypothetical queries, triggers, and a method for computing reservation timespan based on demands on tickets. As verified by our preliminary experimental study, our model and methods significantly improve the existing e-ticket systems. To the best of our knowledge, this work is the first effort for reducing both the false availability and holdback rates. It yields a practical approach for providing effective support for customer reservations in e-commerce systems, including but *not limited* to e-ticket systems.

A large number of online ticket booking systems have been developed. We surveyed 30 popular e-ticket systems, including Expedia [2], Orbitz [3], Priceline [4], MakemyTrip.com, Amadeus.net as well as American Airline, Continental, Southwest, United, and US Airways [1]. As remarked in Section 1, these services only support limited customer reservations. Different virtual travel agencies offer tickets with substantially different prices and conditions for the same customer request. We found that ticket prices may vary by as much as 18% across these agencies. Thus it is desirable for customers to make reservations with a locked price beforehand.

There has been work on a variety of aspects of e-ticket systems. The need for e-ticket systems to interact with airline, hotel and payment services was advocated in [12]. An approach for building a virtual travel agency by composing "hotel booking" and "flight booking" services was proposed in [13]. A prototype for a virtual travel agency was developed in [14], based on ontology and semantic web. An enhanced user interface for B2C booking systems was proposed in [15], by means of a virtual intermediate travel agent. There has also been a host of work on generic Web services, notably on Web service compositions (*e.g.,* [10,9]).

However, to our best knowledge, no prior work has studied the false availability and holdback problems associated with existing e-ticket systems.

Airlines have to deal with reservation cancellations and no-show-ups at flight departure on a daily basis. To avoid revenue loss, most airlines compromise cancellations and no-show-ups by over-booking flights, i.e., booking excessive seats above the physical airplane capacity. There has been previous work on optimizing over-booking [16] [17] [18], which attempts to accurately estimate the number of cancellations and no-show-ups. This differs from our work in that our model provides an adaptive reservation timespan for customers to *hold* the tickets with a fixed price. Although over-booking increases seat availability, they do not allow the customers to purchase their tickets with the price they agreed upon after various reservation timespans. Furthermore, our work aims to facilitate the composition of web services for booking a travel package online. Customers can hold (reserve) a ticket while inspecting successive components or legs of the travel package. They can examine each component one by one, reserving a ticket with a fixed price before proceeding to the next component; and finally, they can "optimize" various picks, combine them to get a reasonable composition of the entire package, without worrying about the availability of tickets reserved for previous legs or the hiking-up of the price of those tickets. In contrast, to the best of our knowledge no airlines provide such a functionality. In essence, our reservation model is customer-oriented, i.e., to serve the best interest of customers, while over-booking is airline-oriented, for the best interest of airline business. The latter does not help with composition of online booking web services.

Hypothetical queries have proven useful in a wide range of applications such as decision support, version management, active databases, integrity maintenance and recently XML updates [19,5,20,21,6]. In this work we leverage hypothetical queries to reduce the overhead of transaction management incurred by customer reservations, and capitalize on the implementation technique proposed by [6] to efficiently support query evaluation (Section 2.2).

There is naturally much more to be done. First, to compute reservation timespan one might also want to take into account other information, such as promotion and sale by airlines, beyond what we have considered in Section 2.2. Second, more experiments should necessarily be conducted on real-world data, *e.g.,* real-life patterns of customer arrival and ticket booking. Third, transaction control is far more intriguing and thus deserves a full treatment for e-ticket services that are built via compositions of other services. Finally, it is interesting and practical to investigate the specific need and requirements for providing the reservation functionality for, *e.g.,* finance services.

## References


1. Mitchell, C., Newton, J., Willdorf, N., Bennett, A., Stellin, S.: A to z guide to travel secrets you need to know (2007), `http://www.travelandleisure.com/articles/a-to-z-guide-to-travel-secrets-you-need-to-know`
2. Expedia.com, `http://www.expedia.com`

3. Orbitz, http://www.orbitz.com/
4. Priceline.com, http://tickets.priceline.com
5. Bonner, A.J.: Hypothetical datalog complexity and expressibility. Theoretical Computer Science 76, 3–51 (1990)
6. Griffin, T., Hull, R.: A framework for implementing hypothetical queries. In: SIGMOD (1997)
7. Sistla, A.P., Wolfson, O.: Triggers on database histories. IEEE Quarterly Bulletin on Data Engineering, Special Issue on Active Databases 15, 48–51 (1992)
8. Ramakrishnan, R., Gehrke, J.: Database Management Systems, McGraw-Hill Higher Education. McGraw-Hill, New York (2000)
9. Berardi, D., Calvanese, D., Giacomo, G.D., Hull, R., Mecella, M.: Automatic composition of transition-based semantic web services with messaging. In: VLDB, Trondheim, Norway (2005)
10. Berardi, D., Giacomo, G.D., Lenzerini, M., Mecella, M., Calvanese, D.: Synthesis of underspecified composite e-services based on automated reasoning. In: ICSOC (2004)
11. Willig, A.: A short introduction to queueing theory, http://www.tkn.tu-berlin.de/curricula/ws0203/ue-kn/qt.pdf
12. Haas, H.: Web service use case: Travel reservation. W3C (2002)
13. Pistore, M., Roberti, P., Traverso, P.: Process-level composition of executable web services: "on-the-fly" versus "once-for-all" composition. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, Springer, Heidelberg (2005)
14. Zaremba, M., Moran, M., Haselwanter, T.: Applying semantic web services to virtual travel agency case study. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, Springer, Heidelberg (2006)
15. Malizia, A.: Adding flexibility to b2c booking systems using a virtual intermediate travel agent. In: HVL/HCC (2005)
16. Leder, K.Z., Spagniole, S.E., Wild, S.M.: Probabilistically optimized airline overbooking strategies, or "anyone willing to take a later flight?!". The UMAP Journal (2002)
17. Klophaus, R., Pölt, S.: Airline overbooking with dynamic spoilage costs. Journal of Revenue and Pricing Management (2007)
18. Lawrence, R.D., Hong, S.J., Cherrier, J.: Passenger-based predictive modeling of airline no-show rates. In: KDD 2003. Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM Press, New York (2003)
19. Balmin, A., Papadimitriou, T., Papakonstantinou, Y.: Hypothetical queries in an olap environment. In: VLDB (2000)
20. Fan, W., Cong, G., Bohannon, P.: Querying XML with update syntax. In: SIGMOD (2007)
21. Gabbay, D.M., Giordano, L., Martelli, A., Olivetti, N.: A language for handling hypothetical updates and inconsistency. Journal of IGPL 4, 385–416 (1996)